FedDW: A TOOL FOR QUERYING FEDERATIONS OF DATA WAREHOUSES

Architecture, Use Case and Implementation

Stefan Berger and Michael Schrefl
Department of Business Informatics – Data & Knowledge Engineering
University of Linz, Altenberger Str. 69, Linz, Austria

Keywords: Data integration systems, Heterogeneous data sources, Federated data warehouses, Distributed query processing, Software engineering.

Abstract: Recently, Federated Data Warehouses – collections of autonomous and heterogeneous Data Marts – have become increasingly attractive as they enable the exchange of business information across organization boundaries. The advantage of federated architectures is that users may access the global, mediated schema with OLAP applications, while the Data Marts need not be changed and retain full autonomy. Although the underlying concepts are mature, tool support for Federated DWs has been poor so far. This paper presents the prototype of the "FedDW" Query Tool that supports distributed query processing in federations of ROLAP Data Marts. It acts as middleware component that reformulates user queries according to semantic correspondences between the autonomous Data Marts. We explain FedDW's architecture, demonstrate a use-case and explain our implementation. We regard our proof-of-concept prototype as a first step towards the development of industrial strength query tools for DW federations.

1 INTRODUCTION

Data Warehouses (DWs) are complex and very large databases optimized for the support of analytical decision making. Typically, a DW collects and consolidates all data in the scope of the enterprise from disparate sources. More specific data repositories, called *Data Marts*, may be derived from the DW to deliver some subset of data to particular user groups (Inmon, 2005). The integration of DW systems often becomes necessary due to business cooperations or mergers and acquisitions between large-scaled companies.

Federated Data Warehouses (FDWs) are data integration systems, allowing transparent access to the heterogeneous schemas of autonomous Data Marts. Such systems provide a global, "mediated" schema by storing semantic correspondences – called *matches* – among the source schemas. At query time, the FDW uses the matches to rewrite user queries.

Schema integration is the necessary prerequisite for data integration systems to work. The two challenges of schema integration are *schema matching* and *data matching*. The former means finding correspondences among elements of autonomous schemas – e.g., their attributes – whereas the latter aims at identifying the real-world entities referred to by the tuples (Doan and Halevy, 2005).

For example, consider the telecommunications market and assume that two competing mobile network providers – we call them "Red" and "Blue" – have agreed upon sharing their DW data for strategic decision making. This cooperation offers advantages to both partners since the knowledge base for their strategic business decisions broadens beyond the original organizational boundaries. Therefore, it becomes much easier for the two providers to analyze questions such as "How much was last year's turnover, grouped by month and product?", or "Which service plans are most popular among 21–30 year old customers?", etc.

Despite its obvious benefits from the business perspective, the integration of autonomous Data Warehouses is difficult and laborious for both, technical and organizational reasons. Technically, numerous heterogeneities among the schemas and data need to be resolved. From the organizational viewpoint, DW access is often restricted to ensure the privacy of confidential data. Thus, the complete physical combination of the autonomous DWs – which would be the easiest solution – is often impractical, especially for large-scaled systems. Such a situation commonly oc-

Berger S. and Schrefl M. (2009).
FedDW: A TOOL FOR QUERYING FEDERATIONS OF DATA WAREHOUSES - Architecture, Use Case and Implementation.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 113-122
DOI: 10.5220/0001985601130122

curs in practice, sometimes even among the divisions of a single company (Kimball, 2002).

In our example, Red and Blue decide to share their Data Marts about phone connections (Figs. 1 and 2). Since both companies have developed their DW schemas independently, numerous heterogeneities exist among the two Data Marts, although they represent the same application domain. Obviously both fact tables can only be unified after some previous transformations of their schemas and their data. For example, Red has modelled two measures, tn_tel and tn_misc for turnovers generated with telephony and miscellaneous services, respectively (Fig. 1), whereas Blue has chosen a single measure named turnover (Fig. 2). Section 4 will detail the two example Data Marts and demonstrate our approach to the integration problem.

red::connections (date_hr: date [date_hr], cust_sscode: customer [cust_sscode], product: products [product]; duration, tn_tel, tn_misc)

date_hr	cust_sscode	product	duration	tn_tel	tn_misc
01.01.08 08:00	1234010180	MobCom	58	11.60	1.00
01.01.08 08:00	1234010180	HandyTel	20	5.50	2.22
03.01.08 08:00	4567040871	FiveCom	250	24.10	4.50
30.06.08 08:00	9876543210	HandyTel	130	14.20	4.00
30.06.08 10:30	9876543210	FiveCom	28	7.10	0.80

red::date (date_hr ↦ date ↦ month ↦ year)

date_hr	date	month	year
01.01.08 08:00	01.01.08	01/08	2008
03.01.08 08:00	03.01.08	01/08	2008
30.06.08 08:00	30.06.08	06/08	2008
30.06.08 10:30	30.06.08	06/08	2008

red::products (product)

product	prod_name	regular_fee
MobCom	Mobile Comm., Ltd.	0.10
HandyTel	Handy Telephone Co.	0.08
FiveCom	Five Communications, Inc.	0.12

red::customer (cust_sscode ↦ contract_type)

cust_sscode	name	contract_type	base_fee
1234010180	Doe, John	SparCom	15.0
4567040871	Stone, Jack	SparCom	15.0
9876543210	Miller, Alice	FlatRate	45.0

Figure 1: Fact and dimension tables of company “Red”.

Assuming that Red’s management wants to access the integrated view on Blue’s and its own data as soon as possible, current DW approaches fall short. Due to the autonomy of both companies, the two “connections” Data Marts cannot be physically integrated unless Red and Blue would create a common central authority. As the review of related work indicates (see Section 2), there is need for approaches and tools capable of DW integration on the logical schema level.

Our previous work has studied Data Mart integration from the conceptual perspective. We introduced the *Dimension Algebra* and *Fact Algebra* with conversion operators for multi-dimensional schemas and data, addressing all heterogeneities identified in (Berger and Schrefl, 2006). Mappings formulated with these operators overcome structural and data heterogeneity among the logical schemas of autonomous Data Marts (Berger and Schrefl, 2008).

In this paper we present the prototype of *FedDW*, a Query Tool for federations of ROLAP Data Marts. The tool allows its users to match autonomous Data Marts with *SQL-MDi* statements – that implement the Dimension Algebra and Fact Algebra operators – and formulate SQL queries over the global schema. When evaluating end user queries, FedDW generates a query plan over the local Data Marts from the operator tree.

The main contribution of our paper is the proof-of-concept implementation of a query tool for Data Mart federations. To the best of our knowledge, FedDW is the first query tool that combines query answering in federated systems with multi-dimensional data integration. Our prototype represents a useful first step towards the development of industrial strength query tools for Federated Data Warehouses.

The paper is structured as follows. Sect. 2 reviews previous approaches of DW integration. In Sections 3 and 4 we summarize FedDW’s functionality and illustrate its use, while Section 5 sketches the tool’s implementation. Finally, Section 6 concludes the paper.

2 STATE OF THE ART

Integration of heterogeneous data has been a major challenge for the database community over the past decades (Halevy et al., 2006; Doan and Halevy, 2005). Lately, the more sophisticated problem of Data Warehouse integration has drawn the attention of researchers. In the following subsections we briefly review previous effort made in these directions.

2.1 Multi-system Query Languages

The first approaches towards interoperability of heterogeneous databases have developed *multi-database query languages*. Multi-database systems are “lightweight” federations without a global schema. In these systems, the user is responsible for matching the heterogeneities among data when formulating the query.

Multi-database languages extend standard query languages with conflict resolution features for schema and data heterogeneity. The most useful approach for the Data Warehousing area is nD-SQL. The nD-SQL language introduces powerful schema transformations – e.g., variables ranging over attribute names and other meta-data – and even supports multi-dimensional data (Gingras and Lakshmanan, 1998). Aggregation hierarchies in dimensions cannot be represented, though, which seems too restrictive in most

blue::connections (date: dates [date], customer: customers [customer_id], product: products [product], promotion: promotions [promo], category; dur_min, turnover)

date	customer	product	promotion	category	dur_min	turnover
01.01.08	1234010180	MobCom	noPromo	tn_tel	17	2.55
01.01.08	1234010180	MobCom	Fall 2008	tn_tel	23	8.50
01.01.08	1234010180	MobCom	Winter 2007	tn_tel	9	4.50
30.06.08	9876090875	HandyTelCo	noPromo	tn_tel	15	1.20
25.08.08	6785041070	HandyTelCo	Christmas 2008	tn_tel	50	4.77
25.08.08	6785041070	HandyTelCo	Christmas 2008	tn_misc	0	2.70

blue::dates (date ↦ month ↦ quarter ↦ year)

date	month	quarter	year
01.01.08	01/08	Q1/08	2008
02.01.08	01/08	Q1/08	2008
30.06.08	06/08	Q2/08	2008
25.08.08	08/08	Q3/08	2008

blue::products (product)

product	prod_name
MobCom	Mobile Comm., Ltd.
HandyTelCo	Handy Telephone Co.
MobileCall	Mobile Calling, Inc.
WirelessCom	WirelessCom, Inc.

blue::customers (customer ↦ contract_type)

customer_id	cust_name	age_grp	contract_type	base_fee
1234010180	Doe, John	21–30	FullCom	15.0
6785041070	Tanner, Frank	31–40	2for0	20.0
9876090875	Miller, Alice	31–40	FullComLite	10.0

blue::promotions (promo ↦ promo_type)

promotion	promo_type
noPromo	No promotion
Winter 2007	Radio spot
Fall 2008	Advertisement
Christmas 2008	TV spot

Figure 2: Fact and dimension tables of company “Blue”.

practical scenarios. Other prominent examples of multi-database languages are Schema-SQL (Lakshmanan et al., 2001) and MSQL (Grant et al., 1993).

Only few approaches have addressed extensions of standard SQL for OLAP applications in distributed Data Warehouses. For example, SQL_M formally defines an OLAP data model and introduces a set of transformation operators. The SQL_M language supports irregular hierarchies in dimensions of standalone Data Warehouses – e.g., multiple aggregation paths. Although the approach allows the user to include additional XML data extending the query scope, SQL_M does not support matches between multiple Data Mart schemas (Pedersen et al., 2002).

In contrast, the SQL-MDi language allows OLAP queries across several autonomous Data Marts. SQL-MDi introduces sets of conversion operators that resolve heterogeneities among multi-dimensional schemas and data, including aggregation hierarchies (Berger and Schrefl, 2006). At query time, the FDW uses the conversion operators to translate distributed data to the global schema, and it computes the query result from the virtual global cube. The semantics of SQL-MDi has been formally defined with the so-called Dimension Algebra and Fact Algebra, that are based upon closed set operations like the Relational Algebra (Berger and Schrefl, 2008).

2.2 Integration of Data Warehouses

Data Warehouse integration is complex because their schemas conform to the multi-dimensional data model. This means that so-called “data cubes” store *facts* in measure attributes that are categorized by the attributes of several *dimensions*. The dimensions, in turn, may be organized in hierarchies of aggregation levels. Thus, matchings of multi-dimensional schemas have to consider (1) the attributes in facts and dimensions, (2) the dependencies between facts and dimensions, and (3) the aggregation hierarchies.

Dimension integration – a sub-problem of multi-dimensional data integration – has been addressed by some recent projects. For example, “DaWaII” is a visual tool that allows the user to specify mappings between dimensions and check their compatibility (Cabibbo and Torlone, 2005; Torlone and Panella, 2005). In order to automatically discover semantic matches among dimension schemas, the approach of (Banek et al., 2007) combines structural schema comparison with linguistic heuristics.

In contrast, fact integration has not received much attention yet. Although integration techniques developed for databases – e.g., (Zhao and Ram, 2007) – probably also apply for Data Warehousing, the automatic discovery of matches between facts of autonomous Data Marts remain an open research problem. Thus, the facts of multi-dimensional schemas still have to be matched manually.

In order to reduce complexity, the integration of multi-dimensional data should concentrate on Data Marts instead of complete DWs. Some previous approaches have proposed loosely coupled Data Mart federations without a global schema. For example, (Mangisengi et al., 2003) define an XML-based query language which allows the ad-hoc integration of Data Marts. The XML layer of their approach exchanges meta-data between autonomous schemas and supports some simple transformation of the data. In contrast, (Abelló et al., 2002) define relationships based on structural similarity between Data Marts that enable drill-across queries. Clearly, the disadvantage of such systems is that the user is responsible for repairing all conflicts within the query.

Federations of Data Marts with a mediated global schema – called *Federated Data Warehouses* – are the ideal solution from the user’s point of view. Such an architecture hides schema and data heterogeneity while the participating Data Marts remain autonomous. The FDW manages the mappings between the global and local schemas, which allows the refor-

mulation of user queries and translation of the query results (Berger and Schrefl, 2008). The federated approach isolates the global schema from changes in the Data Marts. Should the autonomous schemas evolve, the global schema remains valid – only the outdated mappings must be changed accordingly.

Algorithms for query processing in Distributed DWs have brought interesting findings that are also relevant for Federated DWs. In particular, two approaches named Skalla (Akinde et al., 2003) and DWS-AQA (Bernardino et al., 2002) have independently introduced the idea of globally replicating consolidated dimensions. The experiments conducted in both these approaches indicate that replicated dimensions improve the overall response time of queries against the global schema. Query answering in such settings is reduced to the retrieval of facts because all dimension data is globally available.

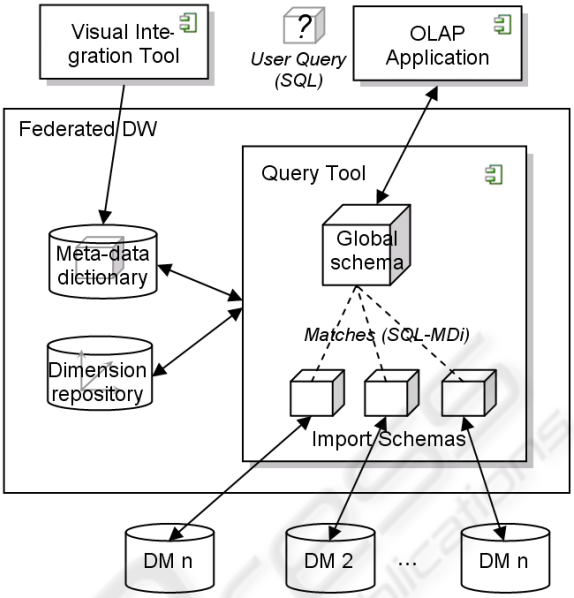

Figure 3: Federated Data Warehouse Architecture.

3 FedDW ARCHITECTURE

"FedDW" is the prototype of a query tool for Federated Data Warehouse systems, as depicted in Fig. 3. In this architecture, the query tool acts as *mediator*: FedDW receives user queries that are formulated over the global schema and rewrites each query to a set of queries against the local Data Marts. Finally, the tool translates the distributed data into the virtual global cube from which it computes the query result.

FDW systems conforming to the architecture depicted above are tightly coupled data integration systems with the global schema defined over several autonomous Data Marts. In this reference architecture, the users access the global schema with OLAP query tools. The intermediate *federation layer* stores the semantic matches between the import schemas of the Data Marts and the global schema, using a *canonical data model*. So-called wrappers manage the communication between the federation layer and the Data Marts since they send the reformulated queries and ship back the answers (Berger and Schrefl, 2008).

FedDW encapsulates the core functionality of the Federated Data Warehouse system. It manages data translation between the autonomous Data Marts and the global schema, according to the semantic matches formulated in the SQL-MDi language. Internally, the FedDW tool expresses the semantic matches as sequences of Dimension Algebra and Fact Algebra operators (Berger and Schrefl, 2008). User queries are answered over the virtual global cube which FedDW computes from the autonomous Data Marts, as specified by the matching operators. Accordingly, the tool consists of two components: the *SQL-MDi parser* and *SQL-MDi processor* (Rossgatterer, 2008).

Two auxiliary components, the *Meta-data Dictionary* and the *Dimension Repository*, complement the functionality of the FDW. The former improves the usability of the system because it not only keeps track of the import schemas for each autonomous Data Mart, but it also saves the semantic matches persistently. The latter increases performance of user query answering by providing local copies of the dimensions in the Data Marts, as proven by the experiments of (Bernardino et al., 2002; Akinde et al., 2003).

The *Visual Integration Tool (VIT)*, which assists in populating the Meta-data Dictionary, is an important supplementary utility in the FDW architecture (cf. Figure 3). The VIT allows the user to define semantic matches between Data Marts and the global schema using UML-based diagrams, and store the matches persistently as SQL-MDi statements. At query time, the FedDW tool automatically retrieves the SQL-MDi code of the matches from the Meta-data Dictionary. The FedDW user may concentrate on OLAP querying without having to bother about conflict resolution. Thus, the usability of FedDW improves considerably.

Currently, the FedDW Query Tool prototype offers the following functionality:

- FedDW's user interface accepts an SQL-MDi statement and an OLAP query – formulated in SQL – as input. The SQL-MDi statement resolves conflicts among the autonomous Data Marts by specifying a global schema and matching it against the Data Mart schemas. In turn, the SQL OLAP query formulates the user's business

question over the virtual global cube, that adheres to the global schema (Figure 4 gives an example).

- On execution of the query, the prototype computes the virtual global cube as instance of the global schema. It then translates all local data as specified by the semantic matches in the SQL-MDi statement, and executes the SQL query. The user interface displays the current progress of query processing (see Figure 4).
- Finally, the query result pops up in an additional window that allows to drill into the details – facts and dimensions – of the virtual global cube (see Figure 5). In OLAP terminology, such operations are denoted as *drill through*.

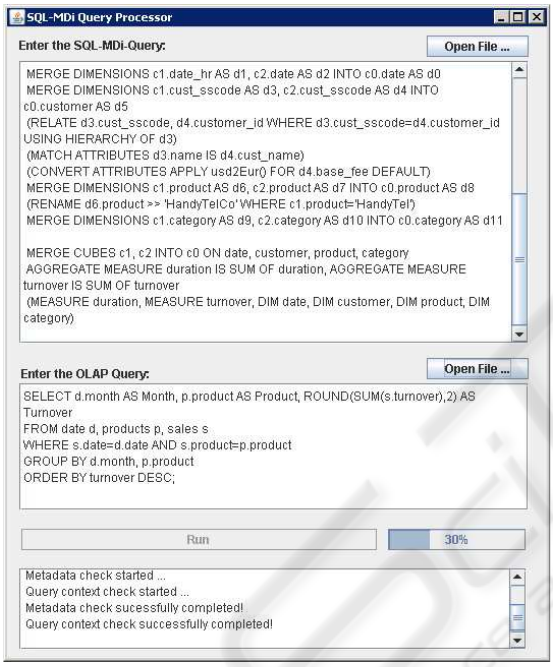

Figure 4: FedDW prototype: Graphical User Interface.

In the next Section we will explain the conflict resolution features of FedDW in detail.

4 USE CASE TOUR OF FedDW

Let us now assume that the management of mobile network provider Red decides to integrate its sales Data Mart with Blue's under a federation. The fact and dimension tables of Red and Blue are shown in Figs. 1 and 2, respectively. In particular, the two Data Mart schemas are briefly characterized as follows:

- Provider "Red" stores the data of turnover within the measures duration, tn_tel and tn_misc, categorized by the dimensions date, customer and products (see Figure 1). The aggregation levels of the dimensions are given in brackets. As monetary unit for the measures Red's schema uses Euros.

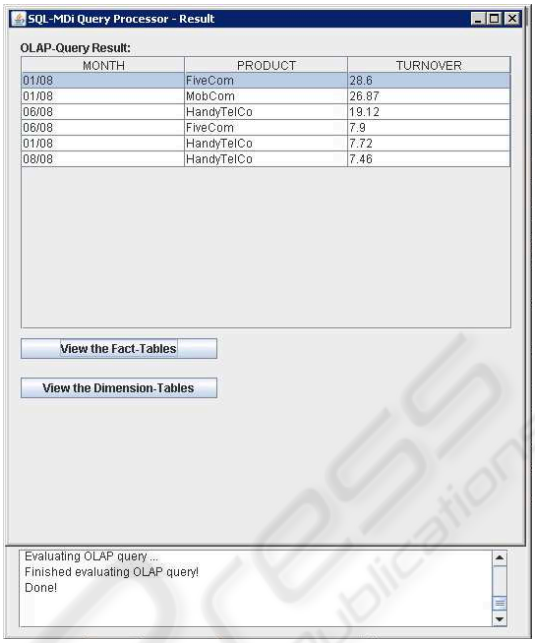

Figure 5: FedDW prototype: View Results.

- "Blue" stores its turnover data within the measures duration and turnover, structured by the dimensions date, customer, product, promotion and category (see Figure 2). Again, the aggregation levels of these dimensions are given in brackets. Blue's schema uses US-Dollars as monetary unit.

FedDW supports numerous strategies for resolving heterogeneities among the facts and dimensions of autonomous Data Marts. Tables 1 and 2 list the conflicts identified in (Berger and Schrefl, 2006) and describe the corresponding resolution techniques employed in FedDW. The two tables also map the resolution techniques to the relevant SQL-MDi operators.

Using FedDW, the Data Marts of Red and Blue may be integrated with the SQL-MDi code listed in Figure 6. This statement matches the local facts and dimensions with the global schema, which is very similar to Red's schema. Figure 7 depicts the fact table of the virtual global cube, as specified by the sample SQL-MDi statement. For brevity, the figure omits the global dimension tables (except customer).

The SQL-MDi language uses three main clauses for defining the global schema and semantic matches: (1) DEFINE [GLOBAL] CUBE, (2) MERGE DIMENSIONS, and (3) MERGE CUBES. The CUBE clauses specify the attribute structure of both the virtual global cube and the import schemas of the autonomous Data Marts.

Table 1: Conflict resolution strategies of FedDW for facts, and corresponding language support.

Conflict addressed	Resolution technique	Relevant SQL-MDi clause of FedDW
–	Import fact table (“cube”)	CUBE operator Example: see lines 1, 5, and 13 in Figure 6
Dimensionality conflicts (Number of DIM references determine cube dimensionality)	Import dimension attributes	DIM keyword Example: see lines 2, 7, 9, and 11 in Figure 6
Overlapping / disjoint measures	Import measure attributes	MEASURE keyword Example: see lines 2, 6, and 7 in Figure 6
Naming conflicts (measures and dimension attributes)	Rename attributes	Operator “–>”; Examples (cf. Figure 6): line 6 (measure attribute) line 9 (dimension attribute)
Schema-instance conflicts (Converts fact context into members of a new dimension)	Merge measures	PIVOT MEASURES ... INTO clause of CUBE Example: see line 3 in Figure 6
Schema-instance conflicts (Generates “contextualized facts” from dimension members)	Split measures	PIVOT MEASURE ... BASED ON clause of CUBE (Not used in sample query.)
Heterogeneous base levels (Corrects domain conflicts among dimension attributes)	Roll-up dimension attributes	ROLLUP clause of CUBE Example: see line 4 in Figure 6
Domain conflicts (measures)	Convert measure domains	CONVERT MEASURES APPLY ... clause of CUBE Example: see line 12 in Figure 6
Overlapping cube cells (fact table extensions)	Join fact tables (“cubes”)	MERGE CUBES ... [set-operation] Example: see line 23 in Figure 6
Overlapping cube cells	Aggregate measure values	AGGREGATE MEASURE clause of MERGE CUBES operator Example: see line 24 in Figure 6

For each dimension of the global schema the MERGE DIMENSIONS clause defines its members and hierarchy from the imported dimensions. Analogously, the MERGE CUBES clause determines how to compute the cells of the global cube from the imported facts.

Within each of its main clauses SQL-MDi defines a number of operators to repair heterogeneous schemas and instances of facts and dimensions (cf. Tables 1 and 2). While the CUBE clause allows to repair heterogeneities among the Data Mart import *schemas*, the MERGE clauses mainly provide operators for the definition of *tuple* matchings and translations in dimensions and facts. Semantic matches between elements of autonomous Data Marts are generally expressed by equal naming. Thus, renaming operations often occur in SQL-MDi, as explained later.

In what follows, we demonstrate how “Red” applies SQL-MDi to resolve the heterogeneities among the Red and Blue sales Data Marts, using the sample statement of Figure 6. For a systematic overview of the available options please refer to Tables 1 and 2. In detail, the sample SQL-MDi statement applies the following conflict resolution techniques:

[Lines 1–12]: In the first part of the query the user specifies the Data Marts' import schemas. Notice that the DEFINE keyword is needed only once. The two CUBE clauses (lines 1 and 5) import the example Data Marts, referencing their measure and dimension attributes with the adequate keywords (MEASURE, DIM). It is mandatory to specify an alias for each cube (AS keyword) to facilitate access to its properties later on.

[Lines 1–4]: The CUBE clause imports Red's Data Mart and performs two transformations. First, line 3 repairs the schema-instance conflict among the two fact tables by merging the existing measures tn_tel and tn_misc into a single turnover measure. This operation generates the *context dimension* “category”. Second, line 4 rolls-up the dimension attribute date_hr to level date, so to match Blue's date granularity.

[Lines 5–12]: The other CUBE clause imports Blue's Data Mart and specifies four transformations. First, the MAP LEVELS clause in line 8 restricts the levels of the date dimension. Only those levels referenced in square brackets are kept in the import schema. Notice that the DIM keyword automatically imports all levels if MAP LEVELS is omitted (as for the dimensions of Red's Data Mart). Second, line 9 renames the dur_min measure attribute. Third, lines 6 and 10 rename dimension attribute customer and level customer_id, respectively. Fourth, line 12 calls the stored procedure usd2Eur() to convert the domain of Blue's turnover measure from US-$ to Euro.

[Lines 2 and 9–11]: Dimensionality of the import schema is determined by the number of explicit references to dimensions of the underlying Data Mart, using the DIM keyword. Recall that schema Blue defines the additional promotions dimension, which is impossible to match with any of Red's dimensions. Thus, dimension promotions is excluded from the import schema of Blue's Data Mart (see line 11, cf. with line 2) by omitting the DIM promotions import specification. This way, the dimensionality conflict

Table 2: Conflict resolution strategies of FedDW for dimensions, and corresponding language support.

Conflict addressed	**Resolution technique**	**Relevant SQL-MDi clause of FedDW**
Heterogeneous hierarchies (Import levels of dimensions referenced by cube import)	Match corresponding levels	MAP LEVELS clause of DIM keyword Example: see lines 8, and 10 in Figure 6
Naming conflicts (level attributes and non-dimensional attributes)	Rename attributes	Operator "–>" (level attributes) Example: see line 10 in Figure 6 MATCH ATTRIBUTES clause of MERGE DIMENSIONS operator (non-dimensional attributes) Example: see line 18 in Figure 6
Overlapping members (dimension extensions)	Merge dimension members	MERGE DIMENSIONS ... [set-operation] Example: see lines 15, 16, 20, and 22 in Fig. 6
Heterogeneous roll-up functions (i.e. hierarchies between members)	Overwrite roll-up hierarchies	RELATE ... USING HIERARCHY OF ... clause of MERGE DIMENSIONS operator Example: see line 17 in Figure 6
Domain conflicts (level attributes and non-dimensional attributes)	Convert attribute domains	CONVERT ATTRIBUTES APPLY ... clause of MERGE DIMENSIONS operator Example: see line 19 in Figure 6
Conflicting values of non-dimensional attributes	Correct attribute values	RENAME clause of MERGE DIMENSIONS Example: see line 21 in Figure 6

among the two Data Marts is repaired; considering the category dimension generated in line 3, both import schemas contain four-dimensional cubes.

[Line 13]: The global schema is not defined immediately. Instead, the GLOBAL CUBE clause is a forward declaration, simply reserving its name and alias for later use (cf. line 23).

[Lines 15–22]: Next, the MERGE DIMENSIONS clauses specify how to populate the global dimensions with tuples ("members"). In our example, the customer dimensions need several directives for conflict resolution: (a) the member hierarchy of Red's customers should override Blue's hierarchy (line 17); (b) the non-dimensional attributes name and cust_name are matched (line 18); and (c) the domain of base_fee is converted from US-$ to Euro (line 19). Moreover, line 21 changes value 'HandyTel' in Red's product dimension to the correct 'HandyTelCo'.

[Lines 23–25]: Finally, the global fact table is determined by the MERGE CUBES clause (line 23), which completes the forward declaration given in line 13. In order to compute the correct values for all measures, line 24 specifies the adequate aggregation function to apply for *overlapping* cube cells.

To examine the global cube defined by the SQL-MDi statement, the user enters an SQL OLAP query as well. In our example, the user has been interested in "How much turnover has our company generated over the last year, grouped by month and product?", as depicted in Figure 4. Upon successful execution of the sample SQL-MDi statement, FedDW returns the virtual global cube. Then it evaluates the SQL OLAP query and displays its result, as shown in Figure 5.

5 FedDW IMPLEMENTATION

Having demonstrated the capabilities of the FedDW tool in our sample use case, the following section will briefly summarize its implementation. It was guided by the following aims: (1) Minimize the design effort through the reuse of well-known software design patterns. (2) Use standards whenever available for FedDW's internal data model in order to maximize the interoperability of FedDW. (3) Support multiple platforms (i.e. databases and operating systems).

5.1 FedDW Query Tool Components

The SQL-MDi parser component of FedDW checks the syntactic and semantic correctness of the SQL-MDi matching expressions, i.e. whether all clauses conform to the EBNF grammar specification of the SQL-MDi language and whether all referenced schema elements exist in the autonomous Data Marts. The parser communicates with the Meta-data Dictionary (cf. subsection 5.2) to verify the query syntax and semantics. If the parser detects errors it returns a description and gives hints on possible reasons.

From the input SQL-MDi statement the parser component generates a data structure called the *operator tree*. It contains the sequence of Fact and Dimension Algebra specified by the SQL-MDi matching expression. The leaf nodes of the operator tree contain the unary algebraic transformations of facts and dimensions, whereas the internal nodes – called "structure nodes" – represent the binary operators, specifying how to merge the intermediate results to the virtual global cube. Upon completion of parsing,

```
1  DEFINE CUBE red::connections AS c1
2    (MEASURE c1.duration, MEASURE c1.tn_tel, MEASURE c1.tn_misc, DIM c1.date_hr
         , DIM c1.cust_sscode, DIM c1.product,
3     PIVOT MEASURES c1.tn_tel, c1.tn_misc INTO c1.turnover USING c1.category)
4    (ROLLUP c1.date_hr TO LEVEL c1.date[date])
5  CUBE blue::connections AS c2
6    (MEASURE c2.dur_min -> duration,
7    MEASURE c2.turnover, DIM c2.date
8      (MAP LEVELS c2.date( [date], [month], [year] )),
9    DIM c2.customer -> cust_sscode
10    (MAP LEVELS c2.customer ( [customer_id -> cust_sscode] , [contract_type] )
          ,
11   DIM c2.product, DIM c2.category)
12    (CONVERT MEASURES APPLY usd2Eur() FOR c2.turnover DEFAULT) )
13 GLOBAL CUBE dw0::sales AS c0

15 MERGE DIMENSIONS c1.date_hr AS d1, c2.date AS d2 INTO c0.date AS d0
16 MERGE DIMENSIONS c1.cust_sscode AS d3, c2.cust_sscode AS d4 INTO c0.customer
       AS d5
17   (RELATE d3.cust_sscode, d4.customer_id WHERE d3.cust_sscode=d4.customer_id
         USING HIERARCHY OF d3)
18   (MATCH ATTRIBUTES d3.name IS d4.cust_name)
19   (CONVERT ATTRIBUTES APPLY usd2Eur() FOR d4.base_fee DEFAULT)
20 MERGE DIMENSIONS c1.product AS d6, c2.product AS d7 INTO c0.product AS d8
21   (RENAME d6.product >> 'HandyTelCo' WHERE c1.product='HandyTel')
22 MERGE DIMENSIONS c1.category AS d9, c2.category AS d10 INTO c0.category AS
       d11
23 MERGE CUBES c1, c2 INTO c0 ON date, customer, product, category
24   AGGREGATE MEASURE duration IS SUM OF duration, AGGREGATE MEASURE turnover
         IS SUM OF turnover
25  (MEASURE duration, MEASURE turnover, DIM date, DIM customer, DIM product,
       DIM category)
```

Figure 6: Example SQL-MDi statement, integrating Red's and Blue's Data Marts.

dw0::sales (date: dates [date], customer: customer [cust_sscode], product: products [product], category; duration, turnover)

date	customer	product	category	duration	turnover
01.01.08	1234010180	MobCom	tn_tel	107	27.15
01.01.08	1234010180	MobCom	tn_misc	58	1.00
01.01.08	1234010180	HandyTelCo	tn_tel	20	5.50
01.01.08	1234010180	HandyTelCo	tn_misc	20	2.22
03.01.08	4567040871	FiveCom	tn_tel	250	24.10
03.01.08	4567040871	FiveCom	tn_misc	250	4.50
30.06.08	9876090875	HandyTelCo	tn_tel	145	15.30
30.06.08	9876090875	HandyTelCo	tn_misc	130	4.00
30.06.08	9876090875	FiveCom	tn_tel	28	7.10
30.06.08	9876090875	FiveCom	tn_misc	28	0.80
25.08.08	6785041070	HandyTelCo	tn_tel	50	4.77
25.08.08	6785041070	HandyTelCo	tn_misc	0	2.70

dw0::customer (cust_sscode ↦ contract_type)

cust_sscode	name	contract_type	base_fee
1234010180	Doe, John	*FullCom*	15.0
4567040871	Stone, Jack	SparCom	15.0
6785041070	Tanner, Frank	2for0	20.0
9876090875	Miller, Alice	FlatRate	45.0

Figure 7: Fact table "sales" of virtual global cube "dw0".

the ordering of operators in the tree is optimized algebraically such that the size of intermediate results is reduced as early as possible (Brunneder, 2008).

FedDW's query processor component receives the operator tree from the SQL-MDi parser and computes the query result over the virtual global cube. All unary transformations in the leaf nodes are traversed "from left to right". Then these intermediate results are combined step by step, according to the structure node operators. The ordering of nodes given in the operator tree remains unchanged (Rossgatterer, 2008).

The critical phase of the query processing algorithm is the generation of an optimal query plan. This means that the processor reformulates the original user query into a sequence of queries over the autonomous Data Marts. If the operator tree refers to dimensions that exist in the Dimension Repository (cf. Section 3), the processor eliminates the according sub-queries from the query plan and instead reads the intermediate results from the repository. Thus, the amount of data shipped between the autonomous DW systems is reduced, as shown by (Bernardino et al., 2002; Akinde et al., 2003).

The implementation of a language parser is known to be rather mechanical and thus easy to automatize. Several software tools – called *compiler generators* – have been developed that enable the automatized generation of parsers. FedDW's parser component has been generated with the JavaCC framework, based upon the formal description – the grammar – of the underlying SQL-MDi language, augmented with so-called production rules (Brunneder, 2008).

In order to optimize the runtime performance and extensibility of the parser and processor components, we applied several well-known software design patterns. As far as the parser is concerned, the generation of the operator tree from the SQL-MDi statement conforms to the *Visitor* design pattern. It consists of (1) a hierarchy of classes representing the nodes of some data structure to be traversed, and (2) the visitor class. Each node class provides an `Accept()` method with the visitor class as input parameter. Moreover, the visitor class contains a corresponding `visitElement()` method for each `Accept()` method of the node classes (Gamma et al., 1995). The JavaCC framework automatically applies the Visitor pattern in the source code it generates (Brunneder, 2008).

The operator tree itself is implemented as *Abstract Syntax Tree* (AST) data structure. An AST represents the syntax of some language string within the nodes of a tree, whereby each node denotes a construct of the language (Reilles, 2007). By combining the Visitor pattern with the AST structure we cleanly separate the implementation of the parser from the definition of the underlying query language. Thus, neither syntax changes nor the introduction of new operators affect the existing implementation. Consequently, the effort necessary to support language extensions is kept minimal (Brunneder, 2008).

Finally, the processor component employs the *Iterator model* for the implementation of the Fact and Dimension Algebra transformation operators. According to the Iterator model, every operator class processes a single tuple instead of a data block per call. Moreover, every operator class works in isolation; this means that it must not communicate with other operator classes nor the overall query plan to compute its result (Graefe and McKenna, 1993). The Iterator pattern offers two advantages. First, it allows to parallelize the traversal of the operators in the tree nodes. Second, the set of operators available in the underlying query language and algebra can easily be extended (Rossgatterer, 2008).

In order to maximize FedDW's compatibility with common databases, its internal data model complies to CWM, the Common Warehouse Metamodel (Medina and Trujillo, 2002). Our FedDW prototype supports the Oracle and Microsoft SQL Server 2005 database systems. Both the parser and processor components are implemented in Java. Therefore, FedDW runs on multiple operating systems.

5.2 Auxiliary System Components

The *Meta-data Dictionary* of the FDW architecture represents the system's schema catalogue. It stores current snapshots of (1) the global schema, (2) each of the local schemas in the autonomous Data Marts, and (3) the semantic matches specified in the SQL-MDi language. The Meta-data Dictionary offers two advantages to the "FedDW" query tool. First, the SQL-MDi processor component may retrieve the semantic matches from the dictionary, such that the details of schema matching and data translation remain transparent to the users. Otherwise the users would be responsible for matching the schemas manually for each query. Second, it enables the SQL-MDi parser component to check the validity of the schema elements (e.g., dimension names, measure attribute names) referenced by the statement.

As depicted in Figure 3, the *Dimension Repository* complements the FDW architecture. The repository holds copies of the data stored in the dimensions of the autonomous Data Marts, translated into the global schema. This approach reduces the complexity of query processing in the FDW system because fewer sub-queries are needed. In particular, all sub-queries that retrieve and translate the dimension data from the autonomous Data Marts can be eliminated from the query plan. The conceptual idea behind the Dimension Repository is motivated by the Skalla (Akinde et al., 2003) and DWS-AQA approaches (Bernardino et al., 2002), as mentioned in subsection 2.2.

6 CONCLUSIONS

In this paper we demonstrated the practical realization of federated Data Warehouse concepts, introducing the "FedDW" Query Tool. We explained the features of its prototype and demonstrated its use with a realistic scenario. The FedDW prototype together with an example scenario and instructions is available for download on our website http://www.dke.jku.at.

Soon we will release the prototype of the Visual Integration Tool as well. As mentioned, the VIT administrates the Meta-data Dictionary of the Federated Data Warehouse approach. It allows the users to persist the semantic matches among autonomous Data Marts and the global schema as SQL-MDi statements. In conjunction, the FedDW Query Tool and Visual Integration Tool offer comprehensive functionality for the integration of autonomous Data Marts.

Our first experiments with FedDW showed reasonable performance and good usability. The viability of the FedDW approach depends on its performance with large data sets. This is subject of our future work.

ACKNOWLEDGEMENTS

We thank Wolfgang Brunneder and Thomas Rossgatterer for implementing the parser respectively the processor components of FedDW.

REFERENCES

Abelló, A., Samos, J., and Saltor, F. (2002). On Relationships Offering New Drill-across Possibilities. In Theodoratos, D., editor, *DOLAP*, pp. 7–13. ACM.

Akinde, M. O., Böhlen, M. H., Johnson, T., Lakshmanan, L. V. S., and Srivastava, D. (2003). Efficient OLAP Query Processing in Distributed Data Warehouses. *Inf. Syst.*, 28(1-2):111–135.

Banek, M., Vrdoljak, B., Tjoa, A. M., and Skocir, Z. (2007). Automating the Schema Matching Process for Heterogeneous Data Warehouses. In Song, I. Y., Eder, J., and Nguyen, T. M., editors, *DaWaK*, volume 4654 of *Lecture Notes in Computer Science*, pp. 45–54. Springer.

Berger, S. and Schrefl, M. (2006). Analysing Multi-Dimensional Data across Autonomous Data Warehouses. In Tjoa, A. M. and Tho, N., editors, *DaWaK*, pp. 120–133. Springer.

Berger, S. and Schrefl, M. (2008). From Federated Databases to a Federated Data Warehouse System. *HICSS*, 0:394.

Bernardino, J., Furtado, P., and Madeira, H. (2002). DWS-AQA: A Cost Effective Approach for Very Large Data Warehouses. In Nascimento, M. A., Özsu, M. T., and Zaïane, O. R., editors, *IDEAS*, pp. 233–242.

Brunneder, W. (2008). Development of an SQL-MDi Query Parser (in German). Master's thesis, Univ. of Linz.

Cabibbo, L. and Torlone, R. (2005). Integrating Heterogeneous Multidimensional Databases. In Frew, J., editor, *SSDBM*, pp. 205–214.

Doan, A. and Halevy, A. Y. (2005). Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, 26(1):83–94.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gingras, F. and Lakshmanan, L. V. S. (1998). nD-SQL: A Multi-dimensional Language for Interoperability and OLAP. In Gupta, A., Shmueli, O., and Widom, J., editors, *VLDB*, pp. 134–145. Morgan Kaufmann.

Graefe, G. and McKenna, W. J. (1993). The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, 1993, Vienna, Austria*, pp. 209–218. IEEE Computer Society.

Grant, J., Litwin, W., Roussopoulos, N., and Sellis, T. K. (1993). Query Languages for Relational Multidatabases. *VLDB J.*, 2(2):153–171.

Halevy, A. Y., Rajaraman, A., and Ordille, J. J. (2006). Data Integration: The Teenage Years. In Dayal, U., Whang, K.-Y., Lomet, D. B., Alonso, G., Lohman, G. M., Kersten, M. L., Cha, S. K., and Kim, Y.-K., editors, *VLDB*, pp. 9–16. ACM.

Inmon, W. (2005). *Building the Data Warehouse*. John Wiley & Sons, New York, 4^{th} edition.

Kimball, R. (2002). *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Datawarehouses*. John Wiley & Sons, 2^{nd} edition.

Lakshmanan, L. V. S., Sadri, F., and Subramanian, S. N. (2001). SchemaSQL: An Extension to SQL for Multidatabase Interoperability. *ACM Trans. Database Syst.*, 26(4):476–519.

Mangisengi, O., Eßmayr, W., Huber, J., and Weippl, E. (2003). XML-based OLAP Query Processing in a Federated Data Warehouses. In *ICEIS (1)*, pp. 71–78.

Medina, E. and Trujillo, J. (2002). A Standard for Representing Multidimensional Properties: The Common Warehouse Metamodel (CWM). In Manolopoulos, Y. and Návrat, P., editors, *ADBIS*, volume 2435 of *Lecture Notes in Computer Science*, pp. 232–247. Springer.

Pedersen, D., Riis, K., and Pedersen, T. B. (2002). A Powerful and SQL-compatible Data Model and Query Language for OLAP. In Zhou, X., editor, *Australasian Database Conference*, volume 5 of *CRPIT*. Australian Computer Society.

Reilles, A. (2007). Canonical Abstract Syntax Trees. *Electronic Notes in Theoretical Computer Science*, 176(4):165 – 179. Proceedings of the 6th International Workshop on Rewriting Logic and its Applications (WRLA 2006).

Rossgatterer, T. (2008). Query Processing in a Federated Data Warehouse System (in German). Master's thesis, University of Linz.

Torlone, R. and Panella, I. (2005). Design and Development of a Tool for Integrating Heterogeneous Data Warehouses. In Tjoa, A. M. and Trujillo, J., editors, *DaWaK*, volume 3589 of *Lecture Notes in Computer Science*, pp. 105–114. Springer.

Zhao, H. and Ram, S. (2007). Combining Schema and Instance Information for Integrating Heterogeneous Data Sources. *Data Knowl. Eng.*, 61(2):281–303.